**Memoria Real**

Gestión: Todo programa debe estar en memoria para que el procesador pueda acceder a los datos de dicho proceso y para que pueda ejecutarse.

La RAM es administrada por el SO (es un vector), está no conoce su contenido.

Requisitos:

* **Reubicación**: poder mover los procesos en diferentes partes de la memoria
* **Protección**: que los procesos no puedan acceder a las áreas de memoria de otro proceso. Sólo el SO debería poder modificar los espacios de otros procesos.
* **Compartición**: que puedan compartirse datos entre procesos (el SO asigna un espacio compartido)
* **Organización física y lógica:** alguna forma de organizar de forma lógica y física la memoria.

**Forma de protección:**

* Registro Base y Limite: en el PCB del proceso se tiene registro del comienzo y el fin de su área de memoria disponible. Sí el proceso quiere acceder a una posición fuera de su rango se genera una interrupción.

|  |  |
| --- | --- |
| **MMU** -> Unidad de administración de memoria, está en el hardware y controla los registros base y límite. Funciona como conversor ya que transforma una dirección lógica a una física. Proporciona mayor velocidad y protección. |  |

**Direcciones lógicas**: sólo las conoce el proceso (las genera el CPU). Es independiente de la ubicación real.

**Dirección relativa:** dirección lógica en que la se expresa según un punto conocido

**Dirección física:** referencia a una verdadera ubicación en memoria.

**Carga y traducción de direcciones:**

* En tiempo de ejecución: el proceso puede cambiar su posición en la memoria continuamente ya que está posición se le asigna en tiempo de ejecución.

**Enlace de bibliotecas compartidas**:

1. carga estática: se cargan en tiempo de comp las biblio que usara el proceso
2. carga dinámica: las bibliotecas se van agregando a la imagen del proceso a medida que éste las requiera.
3. enlace dinámico: las bibliotecas se cargan una vez sola a memoria y los procesos que las quieran usar podran acceder a está en tiempo de ejecución. e.g: las bibliotecas .dll de windows. Este último sirve para ahorrar espacio en la memoria respecto de los otros, sobre todo el de carga estática.

La memoria se encuentra dividida en 2 zonas, para Kernel, para procesos de usuario

**Asignación de memoria para procesos**:

1. Posicionamiento fijo:
   1. la memoria se divide en particiones de tamaño fijo
   2. los procesos se cargan en las particiones disponibles.
   3. Su desventaja es que sí viene un proceso mayor al tamaño de las particiones, este no se podrá cargar en memoria. Además, sufre de fragmentación interna.
2. Asignación dinámica: divide la memoria como la vaya necesitando (mediante estructuras enlazadas por ejemplo).

Esto no tiene fragmentación interna, el problema es que la administración se vuelve más compleja por usar una lista. Pero tiene fragmentación externa ya que cuando terminan los procesos van quedando huecos en la memoria.

Para arreglar esto hay que utilizar la “compactación” (juntar los nodos de la lista):

* Bruta: junta todos los nodos
* Inteligente: va juntando según requiera

**Fragmentación interna:** espacio asignado a un proceso que queda libre pero no puede ser usado por otro proceso diferente. Por lo general ocurre cuando tenemos tam fijos.

**Fragmentación externa:** hay suficiente memoria libre para satisfacer una petición, pero esa memoria no es contigua. Esto último puede solucionarse haciendo una compactación en memoria, lo que va a producir mucho overhead.

**Algoritmos de ubicación:**

FIRST FIT: busca el 1er hueco disponible desde el comienzo de la memoria.

NEXT FIT: busca el 1er hueco disponible desde la posición de la última asignación.

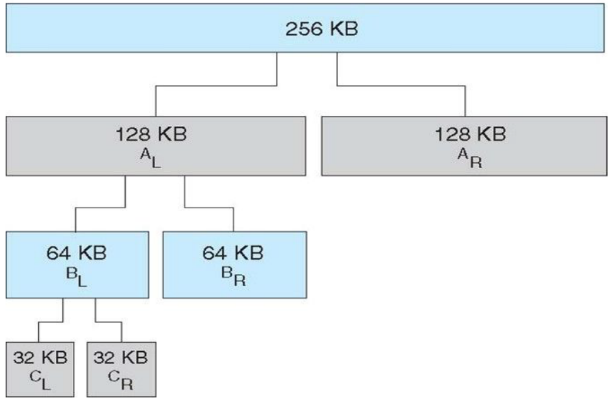
BEST FIT: busca el hueco más chico donde pueda ubicarse el proceso

WORST FIT: busca el hueco más grande donde pueda ubicarse el proceso.

**Buddy System** (sistema de asignación de memoria)

Compensa la desventaja de particionamiento fijo y dinámico. Se le asigna a los procesos tamaños de memoria que sean potencia de dos (2^n). El tamaño que se asigna es el mínimo espacio que sea potencia de dos para que el proceso pueda cargarse. A medida que esas particiones se van liberando se van a juntar para volver a la partición original (la más grande).

Tengo particiones fijas pero de tamaño dinámico. Sufre fragmentación interna y externa pero en menor medida.



**Segmentación:**

* El proceso no necesita estar contiguo en memoria
* El proceso se divide en segmentos de tamaño variable
* Cada segmento representa una parte del proceso (Code, Stack, Heap, Data).
* Sufre fragmentación externa pero menor que los sistemas anteriores y no sufre fragmentación interna ya que los segmentos no son de tamaño fijo.
* Cada proceso tiene su propia tabla de segmentos asignada

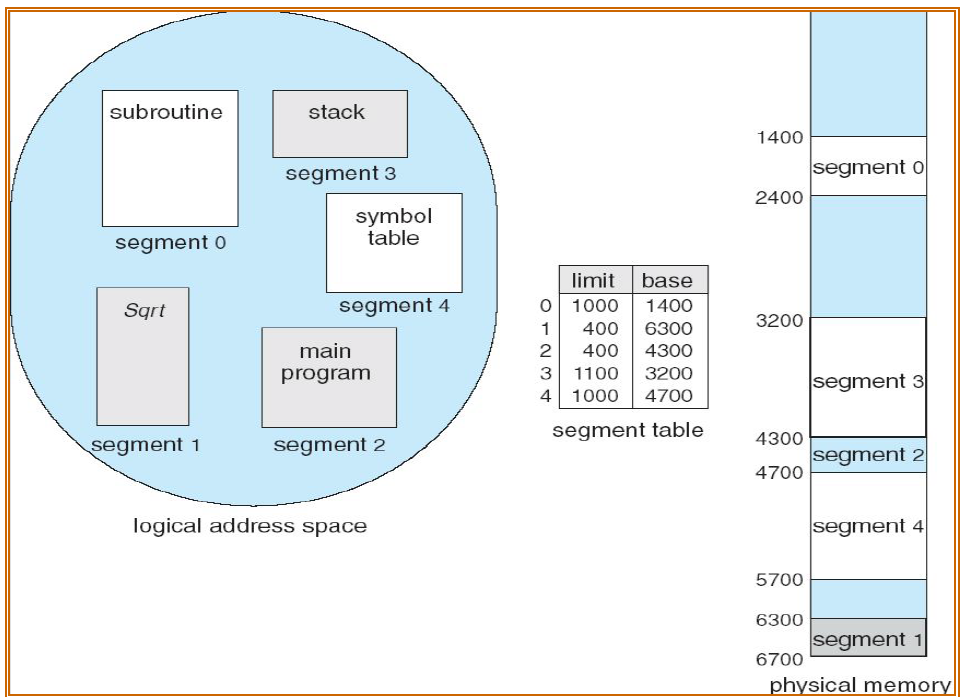
Para calcular la dirección física:

1. buscar el segmento en la tabla de segmentos
2. una vez encontrado, tomo la base y le sumo el offset
3. Esa suma me da la dirección física.

Cantidad máxima de segmentos por proceso = 2^x.

Tamaño máximo de segmento = 2^y

**x** e **y** es la cantidad de bits.



**Protección:** Segmentation fault → cuando el offset + base se sale del segmento seleccionado. Cada segmento puede tener permisos de lectura, escritura y ejecución. (R - X - W)

**Paginación:**

* Proceso no contiguo en memoria
* Procesos y memoria divididos en partes iguales (mismo tamaño)
* La memoria se divide en frames (Tamaño página = Tamaño frame)
* Los procesos se dividen en páginas

Se utiliza la tabla de páginas para manejar las páginas de cada proceso, o sea que hay una por proceso. En esta tabla cada página tendrá su frame asociado.

No hay fragmentación externa pero sí interna ya que puede darse que el proceso ocupe menos del tamaño de una página. En promedio suele ser del tamaño de la página menos 1 byte. EJ: tengo páginas de 4kb y entra 1 proceso de 4097 bytes, este sería el peor de los casos posibles.

Dirección lógica a física: (N° Página + desplazamiento)

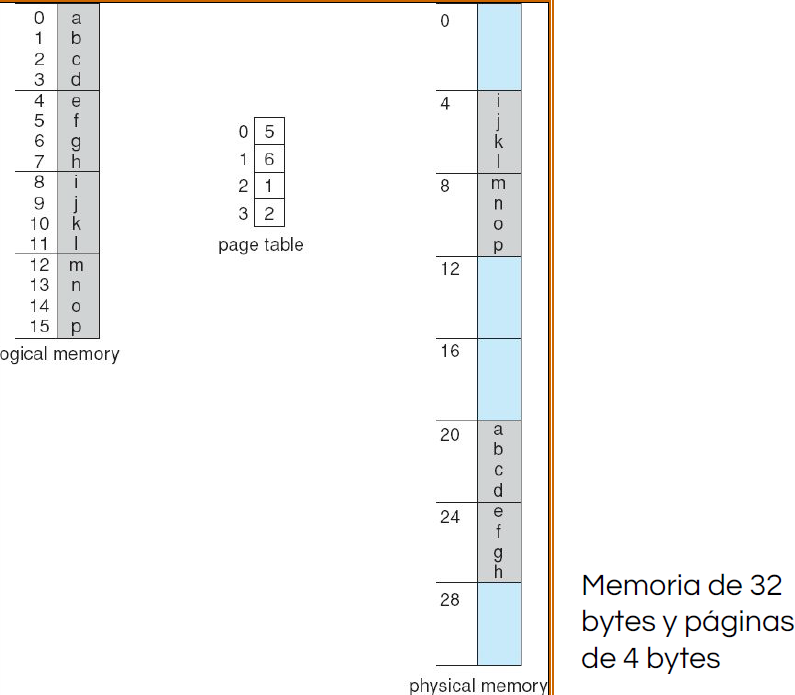
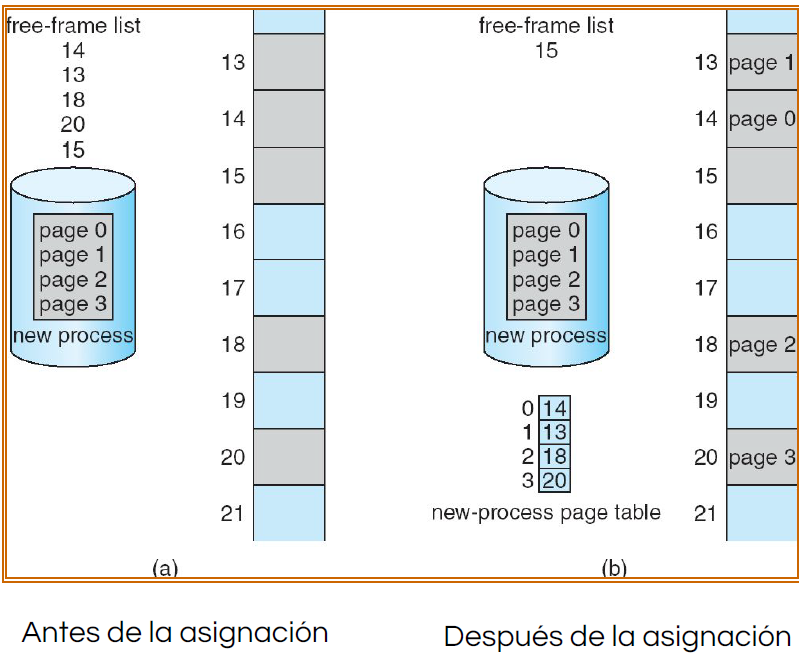
1. busco en número de página en la tabla
2. una vez encontrada la página, saco el número de frame.
3. al número de frame le sumo el offset y ya tengo la dirección física.

Cantidad máxima de páginas por proceso = 2^x; Tamaño de página = 2^y

Esto es igual que en **segmentación.** El tamaño de las paginas es en potencias de 2.

Para llegar a la dirección lógica o física: (Nº Pag/Frame \* tamaño página) + OFFSET

**Protección:** En la tabla de páginas se pueden definir los permisos de cada página.

**Compartición:** es sencilla, se le puede decir a ambos procesos en su tabla de páginas que tienen acceso a una o más páginas en común. **Tabla de marcos libres:** para determinar qué frame de memoria está libre. e.g: Bitmap. 

Memoria de 32 y páginas de 4 (bytes). Antes/después asignación

**Segmentación paginada**

* los segmentos se paginan
* combina ventajas de la segmentación y la paginación.

Única tabla de segmentos y cada segmento tiene una tabla de páginas.

Dirección lógica:

1. busco el segmento en la tabla de segmentos
2. busco la página en la tabla de página de la dirección de memoria que me dio el segmento encontrado
3. con el frame obtenido en la tabla de páginas le sumo el offset y tengo la dirección física.

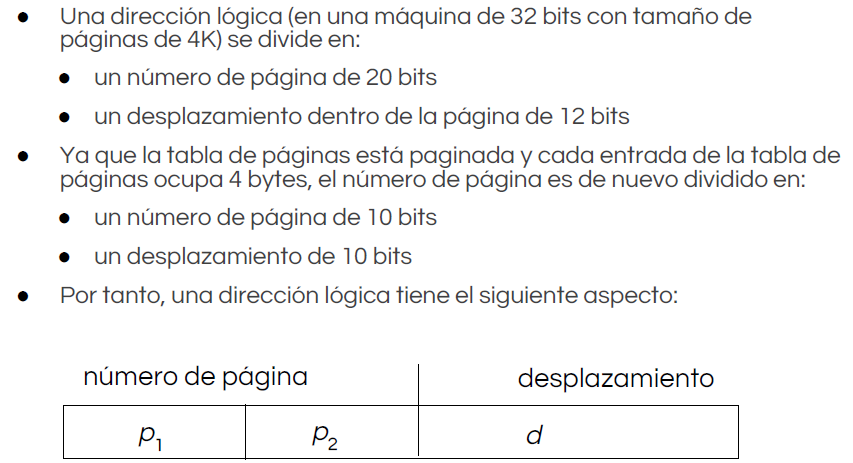
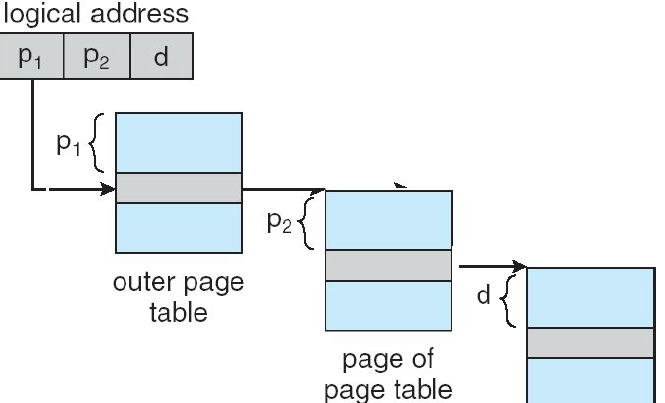
Sigue teniendo fragmentación interna pero muy poca.

**Paginación jerárquica o por niveles:**

tengo una tabla de páginas que tiene punteros a más tablas de páginas (en memoria real). Y todas las tablas de páginas en memoria virtual. **Dirección física**:

1. busco la tabla de paginas
2. busco el número de página
3. busco la dirección física con el número .de marco y offset

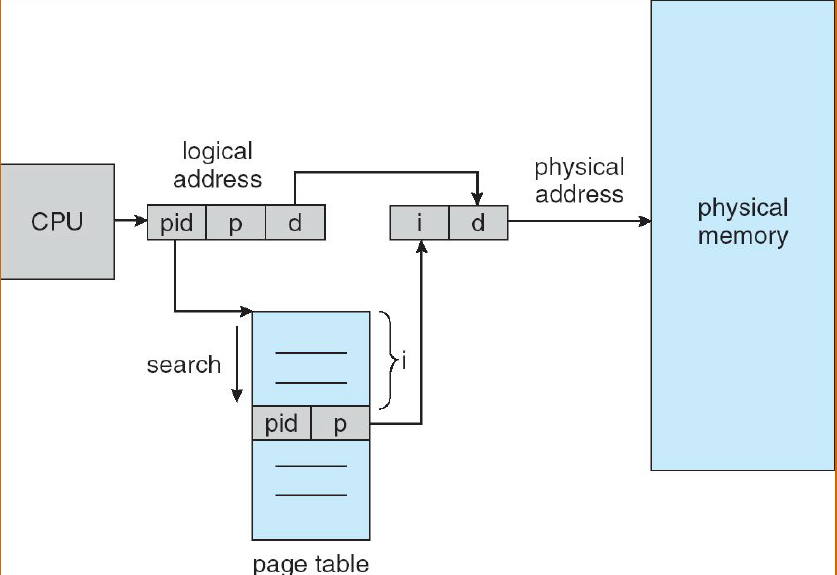
En total tengo 3 accesos a memoria. Me soluciona el problema de tener tablas de páginas muy grandes que no entren en memoria real.

**Tabla de páginas invertida**

* hay una tabla para todos los procesos
* la cantidad de entradas en la tabla es igual al número de marcos en memoria

El problema es la búsqueda ya que sí tengo muchos marcos en la tabla entonces tengo mucho overhead. Esto se mejora con una función de hash: a partir del número de página y el PID me devuelve el número de marco donde está dicha página y me ahorro la búsqueda.



**TLB: Buffer de traducción anticipada**

Memoria pequeña que tiene entradas de la tabla de página (sólo algunas), es un componente de hardware.

Primero busca la página en la TLB, sí no la encuentra (TLB miss) entonces busca en la memoria real y ahi arma la dirección física.

La TLB comienza vacía pero a medida que se use la tabla de páginas está irá pasando las páginas usadas en la TLB para que ya las tenga disponibles. Hay menos accesos a memoria (menos overhead) -> nos ahorra un acceso a memoria.

Cuando se cambia de proceso, la TLB se borra por completo, o sea, que sólo contiene las páginas de un proceso.

**Características:**

* muy rápido pero pocas entradas
* entradas de la TLB
* ASID (identificación de espacio de direcciones)

Se pueden fijar entradas para que no se borren y además a cada entrada se le puede asignar su proceso determinado.

1. Politica de recuperacion o limpieza
   1. Paginación bajo demanda (trae sólo las páginas que necesita)
   2. Paginación adelantada (trae las páginas que necesita más algunas otras que podría llegar a usar)
2. Política de ubicación
   1. con segmentación pura
      1. Algoritmos Best fit, First fit, Next fit y Worst fit.
3. Políticas de reemplazo o sustitución -> eligen que marco podrá ser reemplazado
   1. bloqueo de marcos
   2. algoritmos: Óptimo, FIFO, LRU, Clock, Clock mejorado -> sirven para elegir el marco a reemplazar, me bajan el indice de PF (page fault).

**Algoritmo Óptimo:**

Selección de la víctima: se elige a la página a la que se realizará una referencia en el futuro más lejano. No es aplicable ya que no puedo predecir el futuro.Produce menos Page Fault que los demás.

**Algoritmo FIFO:**

Se elige a la página que hace más tiempo está en memoria (la primera que entró).

**Algoritmo LRU (menos usada recientemente)**

Se elige a la página que hace más tiempo no es referenciada

**Algoritmo del reloj o de segunda oportunidad (Clock)**

Se apunta al proximo marco a reemplazar, pero:

* si bit de uso == 0 → se reemplaza el marco
* sí bit de uso == 1 → bit de uso = 0 y se apunta al siguiente marco y vuelve a preguntar

es cómo un LRU pero sin tanto overhead (parece a FIFO).

**Algoritmo Clock mejorado**

Requiere:

* un puntero al siguiente marco a analizar
* bit de uso (u)
* bit de modificación (m)

Condiciones:

* Sí u=0;m=0 → no accedido, no modificado
* Sí u=1;m=0 → accedido, no modificado
* Sí u=0;m=1 → no accedido, modificado
* Sí u=1;m=1 → accedido, modificado

1. Recorrer los marcos y seleccionar el primero con u=0;m=0
2. Sí no encuentra, recorre los marcos y selecciona el primero con u=0;m=1. A medida que recorre, modifica el bit de uso de cada marco de 1 a 0.
3. Repite el 1er paso y sí es necesario el 2do paso.
4. Política de conjunto residente

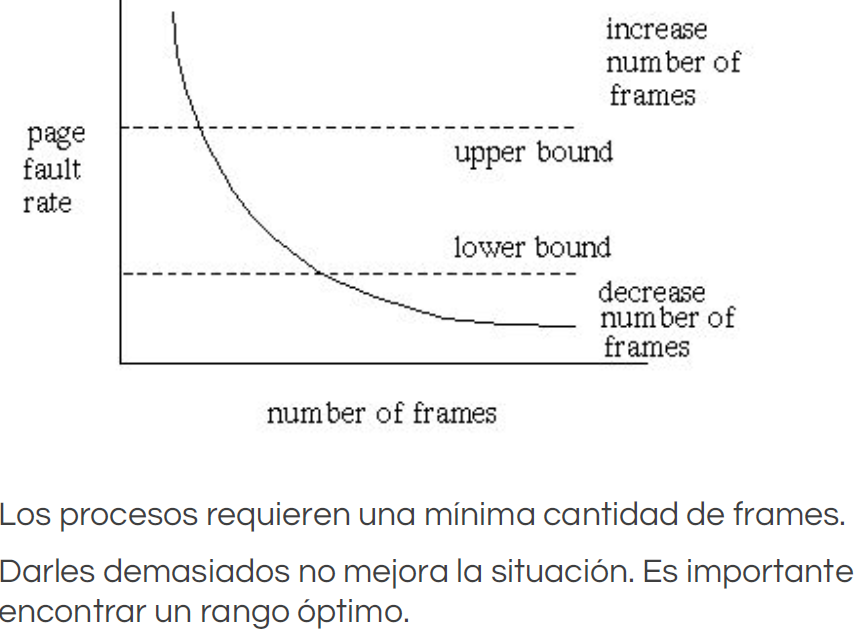
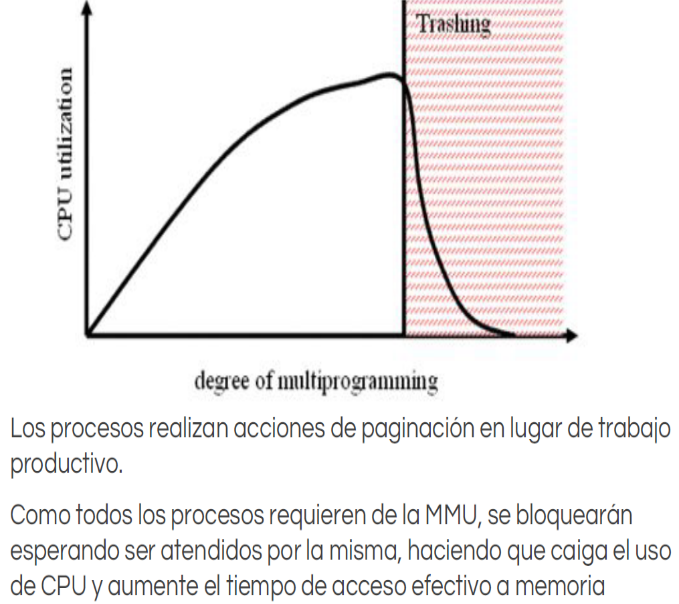
|  |  |  |
| --- | --- | --- |
|  | **Reemplazo Local** | **Reemplazo Global** |
| **Asignación Fija** | Número de marcos asignados a un proceso no varía  Páginas a cambiar son del mismo proceso | No es posible |
| **Asignación Variable** | Número de marcos asignados a un proceso puede cambiar  Páginas a cambiar son del mismo proceso | Las páginas a reemplazar se eligen entre todos los marcos  Cada reemplazo puede afectar a otro proceso |

**Sobre paginación / Trashing**

Puede suceder con: Sustitución local-asignación fija/ Sust global- Asig dinámica

Más procesos en memoria y menos marcos por procesos producen más Page Fault lo cual baja la tasa de uso de CPU y aumenta el grado de multiprogramación. O sea que se invierte más tiempo en el mecanismo de paginación que en el de ejecución del proceso, se producen muchas E/S. Se soluciona agregando más memoria.

Sí las páginas son chicas habrá más Page Fault y sí las páginas son grandes habrá más fragmentación interna.



Problema Solución

**Memoria Virtual**

Usamos la memoria virtual para cargar las páginas de los procesos y luego las voy pasando a memoria. Me sirve cuando quiero correr procesos que no entran en memoria por su tamaño.

Motivaciones: Durante la ejecución del proceso no se usan todas sus páginas o segmentos. Se carga todo el proceso en memoria virtual y en memoria real sólo cargo las páginas que voy necesitando.

**Ventajas:**

* puedo ejecutar procesos más grandes que la memoria real
* además puedo tener más procesos ejecutando a la vez. (Aumento grado de multiprogramación).
* menos restricciones para el programador.

**Definición:** Espacio de memoria secundario (el disco), que puede ser direccionado como si fuese memoria real.

**Mecanismo de búsqueda de páginas:**

1. Se realiza una referencia a memoria
   1. verifico la presencia de la página en memoria
   2. sí el bit ‘p’ está en 0 el MMU lanza INT (page fault).
2. El SO bloquea el proceso
3. Solicitud de página (el disco)
4. Se carga la página en memoria (el SO toma el control)
   1. el bit ‘p’ se actualiza y se pone en 1.
5. El proceso se pone en listo para que pueda ser ejecutado.

Cuando trae la página a memoria real actualiza el frame y el bit de presencia. Ahora tenemos una operación de E/S cuando haya un page fault.

En total tenemos 3 accesos a memoria real y 1 E/S.

bit de modificación: se usa cuando la página se modifica en memoria y al momento de sacarla de memoria tengo que escribir en disco la modificación.

**Eficiencia:**

* requiere más accesos a memoria
* requiere más accesos a disco
* no mejoró el rendimiento para ejecución del proceso

**Principio de proximidad/localidad:**

* Permito que la memoria virtual funcione y que no genere bajo performance.
* Durante un intervalo de tiempo sólo se usan unas páginas de forma activa que se denominan localidad en ese intervalo.
* Temporal: Si en un momento particular esa posición de memoria es referenciada, es posible que en un futuro cercano también vuelva a ser referenciada
* Espacial: Idem pero con las posiciones cercanas a la referenciada.

**Políticas:**

* Asignación: Fija, dinámica
* Sustitución: Local, Global

## File System

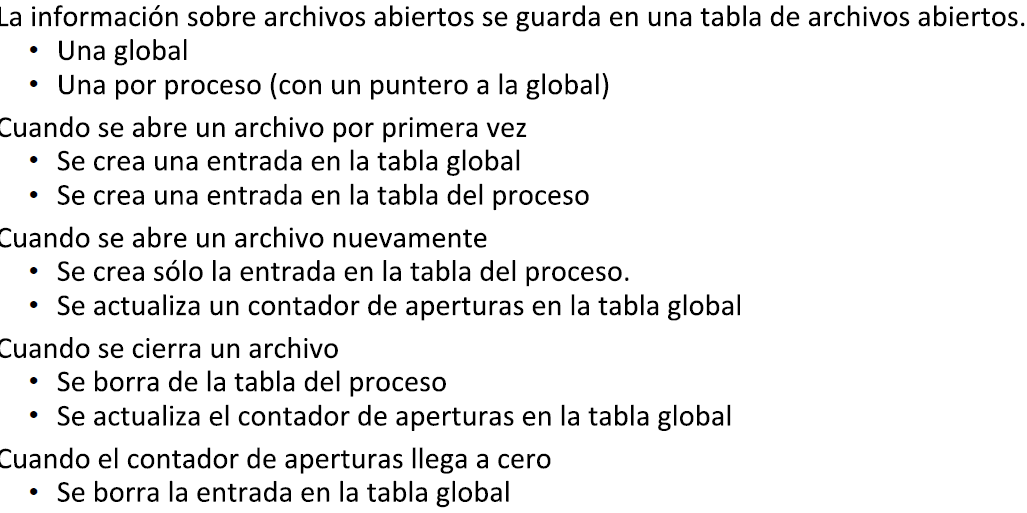
Objetivos:

* almacenar y operar datos
* soporte para varios usuarios
* minimizar posibilidad de pérdida de datos
* maximizar el desempeño del sistema
  + SO: administrar espacio en disco y aprovecharlo
  + Usuario: tiempo de respuesta
* soporte para distintos dispositivos (pendrive - memoria sd - cd)
* garantizar la integridad o coherencia de los datos

Archivo: Conjunto de bytes relacionados entre sí que podemos guardar/almacenar en un dispositivo. Atributos: Nombre, ID, Tipo, Ubicación, Tamaño, Protección, Metadata

Operaciones: crear - abrir - leer - renombrar (mover) - truncar (cambiar tamaño de archivo) - borrar - cerrar - escribir - reposicionar (cambiar valor del puntero dentro del archivo) - **copiar y renombrar (mover) → estas son combinadas.**

Archivos abiertos



Apertura

* Modo de apertura: R - W - A → le tengo que informa al SO para que verifique sí tengo permisos suficientes
* Tabla global de archivos abiertos → contiene los atributos más importantes de cada archivo también tiene la cantidad de veces que se abrió el archivo
* Tabla de archivos abiertos por proceso → para ver los archivos que abrió cada proceso y cada proceso posee un puntero distinto para el archivo los cuales se almacenan en esta tabla

Métodos de acceso

* Secuencial: Un registro después de otro. cinta magnetica - cassete
* Directo: Se puede acceder a cualquier registro sin acceder a los anteriores. SSD – HDD
* Indexado: Se coloca un índice para acelerar búsquedas sobre archivos grandes.
* Hashed: Para agilizar accesos directos. Se utiliza una función de hash para acceder directamente al bloque deseado

Bloqueos / Locks: Permite regular el acceso a un archivo

Evitar que 2 procesos escriban en un archivo simultáneamente.

Evitar que un proceso lea información desactualizada

Mantener la integridad de los archivos

* Compartido (el archivo se comparte con todos) o

Exclusivo (un proceso bloquea un archivo para usarlo el solo)

* Obligatorio (proceso abre archivo y el SO lo bloquea para que no se pueda usar) o Sugerido (el usuario las aplica. El SO deja abrir el archivo, pero no modificarlo sí está en uso)

Tipos de archivos

* Archivos del SO: Regulares, Directorios (carpetas), Links → softlink, Dispositivos - bloque - socket - terminales
* Archivos Regulares:.doc, .pdf, .mp3
* Archivos Ejecutables:.exe, .bat,. con **En linux no hace falta la extensión, sólo permiso de ejecución.**

Rutas de un archivo:

* Ruta absoluta: ruta final del archivo (donde se encuentra almacenado)
* Ruta relativa: ruta que se escribe a partir del directorio actual (Working Dir.)

**Directorio:** lista de archivos u otros directorios

Operaciones: Búsqueda de un archivo - crear archivo - renombrar archivo - borrar archivo - listar un directorio - recorrer FS

Cada bloque del File System puede estar en distintos sectores del disco físico (hardware). La unidad mínima de lectura y escritura en el File System es el **Bloque.** En un bloque sólo hay información de un sólo archivo a la vez.

Partición: División lógica Volumen: Partición con formato

Se posee una tabla de particiones que me da información sobre las particiones. Cuando formateo una partición le digo que File System usara (EXT2, FAT, NTFS).

Estructura de un File System

* Bloque de arranque o booteo → contiene información para saber desde donde leer el SO
* Bloque de control de volumen → información del contenido de archivos
* FCB (File control block) → uno por archivo/directorio, contiene información del archivo. Es de las más importantes.
* Estructura de directorios
  + entradas de directorio: nombre - archivo+atributos o puntero al FCB

En memoria

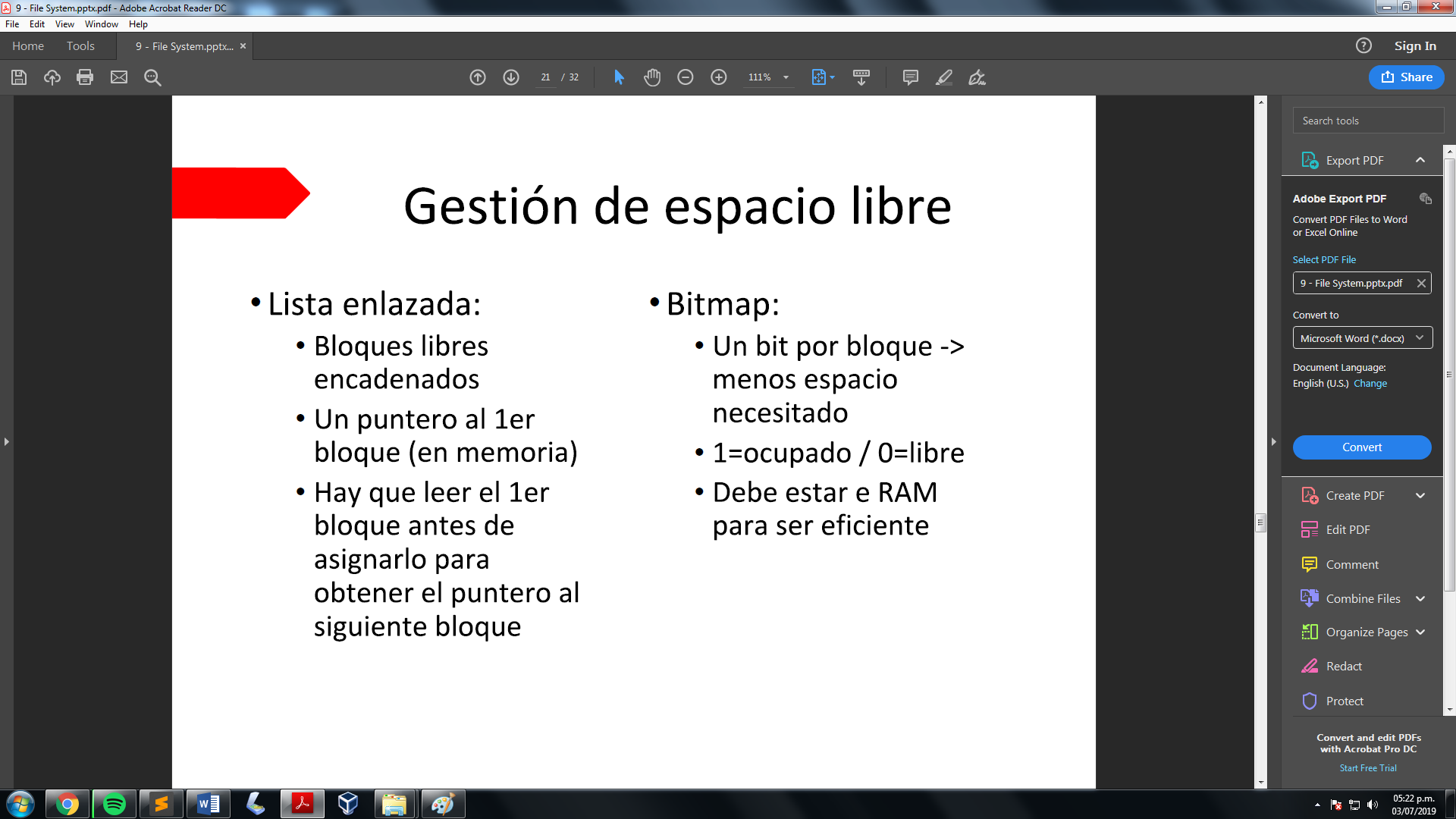
* Tabla de montaje → en que dirección está cada File System
* Estructura de directorios
* Tabla global de archivos abiertos
* Tabla de archivos abiertos por proceso

El File System divide a los archivos en bloques y en el FCB guarda que bloques le pertenece a cada archivo. Los bloques pueden ser de tamaño fijo o variable pero se suele utilizar fijo. Por ende puede sufrir fragmentación interna.

**Métodos de asignación**

1. **Contigua:** un bloque seguido de otro → este esquema sufre fragmentación externa. Es fácil el acceso secuencial y el acceso directo, pero sí quiero agrandar un archivo de tamaño y los bloques siguientes están ocupados, no puedo realizar la operación. La estructura de este sistema consiste en tener una tabla por archivo donde se tiene referencia del bloque inicial y la cantidad de bloques.
2. **Enlazada o encadenada:** Cada bloque contiene al final un puntero al bloque siguiente excepto el último. Resuelve los problemas del esquema anterior. Pero el acceso secuencial debe hacerse recorriendo todos los bloques, o sea, que no permite el acceso directo. Tiene un gran problema, sí pierdo un bloque pierdo la referencia a los siguientes. El tamaño disponible por bloque será: tamaño bloque - puntero. Problema de bloques dispersos=> mucho mov del cabezal
3. **Indexada:** Posee una estructura con los bloques que usa el archivo, los punteros siguen estando pero en la tabla de bloques. Permite acceso secuencial y directo, soluciona los problemas del esquema anterior. Junta ventajas de los modelos anteriores. La desventaja está en su tabla la cual genera más overhead. Problema de bloques disperso=> mucho mov del cabezal

Gestión de espacio libre



**Ejemplo de escritura:**

1. Crear archivo
   1. verificar que haya FCB disponible
   2. crear entrada de directorio
2. Asignarle bloques requeridos
   1. abrir archivo y agregarlo en las listas de archivos abiertos
   2. obtener bloques libres
   3. asignar los bloques al archivo
3. Escribir en el archivo
   1. Escribir los bloques
   2. Actualizar atributos (fechas, tamaño, permisos)
4. Cerrar el archivo
   1. Actualizar lista de archivos abiertos

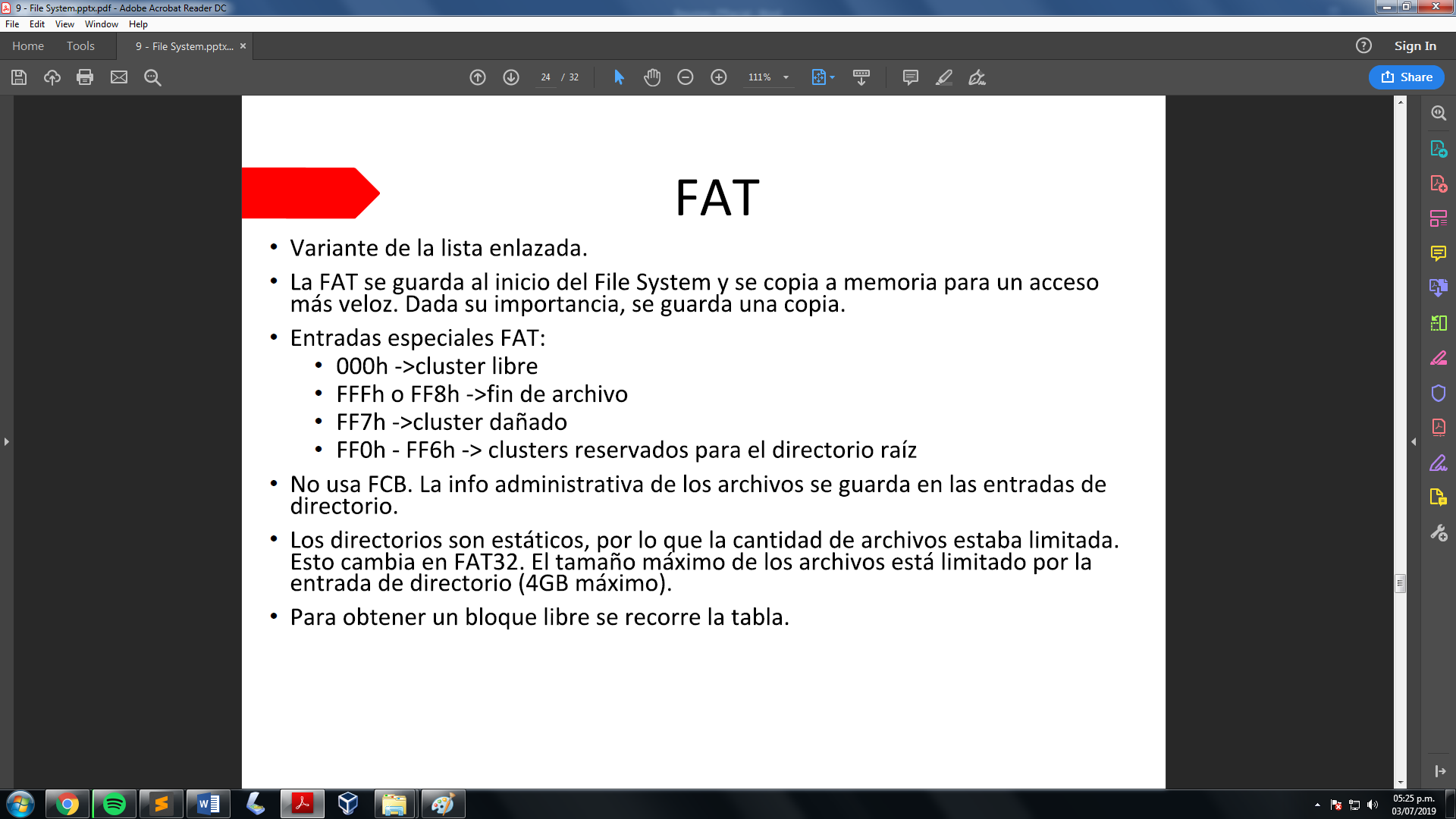
**Recuperación**

* comprobacion de coherencia: herramientas del SO para ver sí el File System no está dañado
* Backups (copia de seguridad)
* Journaling (estructura de registro) → trata las tareas cómo transacciones o sea que sí no se completan en su totalidad no realiza ningún cambio en el File System. Es un método de protección en caso de que ocurra algo inesperado mientras se este modificando el File System.

**Área de Swapping**

* Partición (Linux) → tengo una partición separada llamada linux-swap donde estará la memoria virtual con su File System. Ventaja: es muy simple. Desventaja: Es una E/S lenta y de tamaño fijo
* Archivo (Windows) → pagefile.sys → está toda la información de la memoria virtual. Ventaja: puedo asignarle el tamaño que quiera. Desventaja: es un archivo por ende para modificarlo tengo que llamar al SO y realizar las operaciones

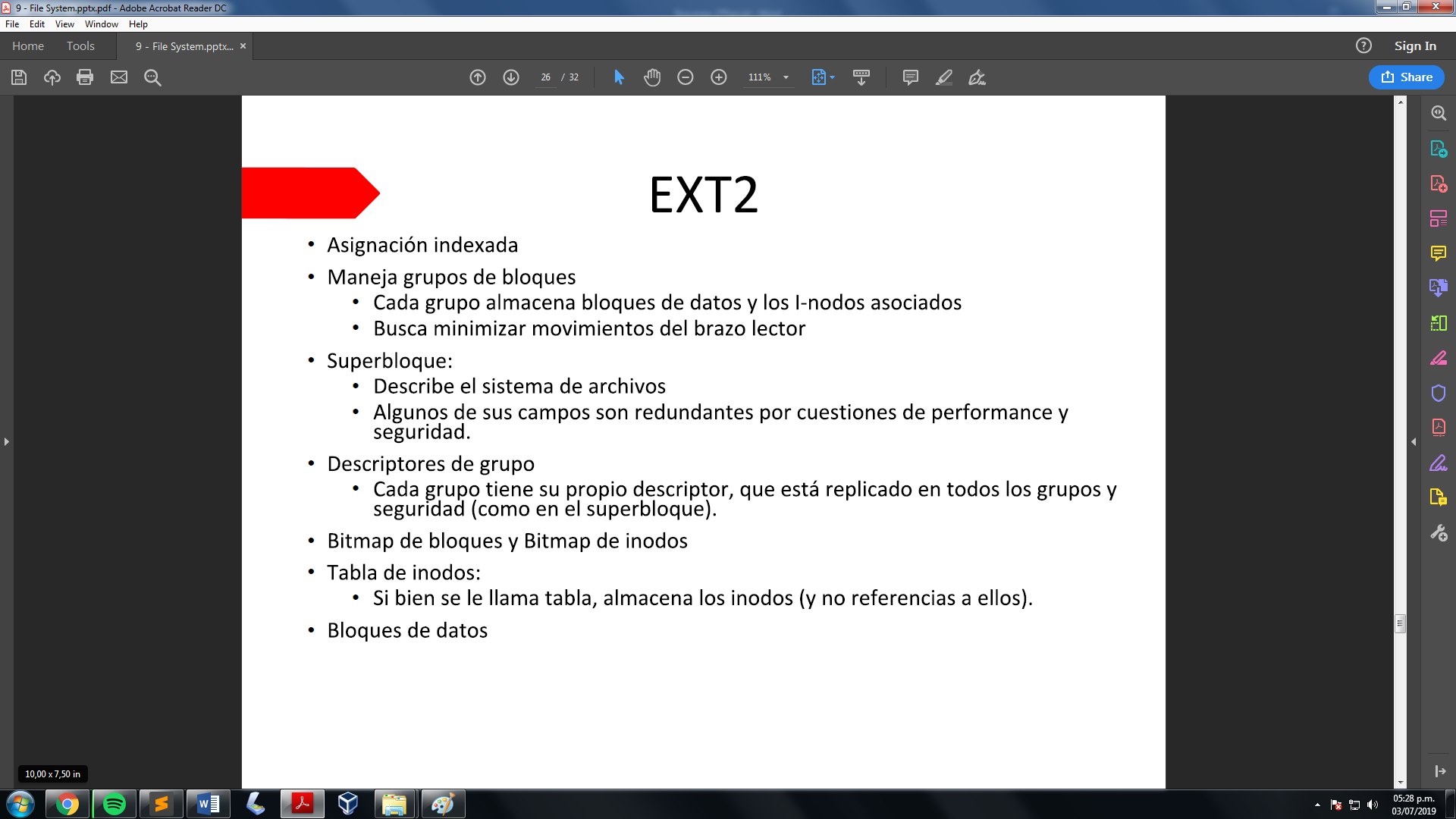
**FAT (file Allocation Table)**



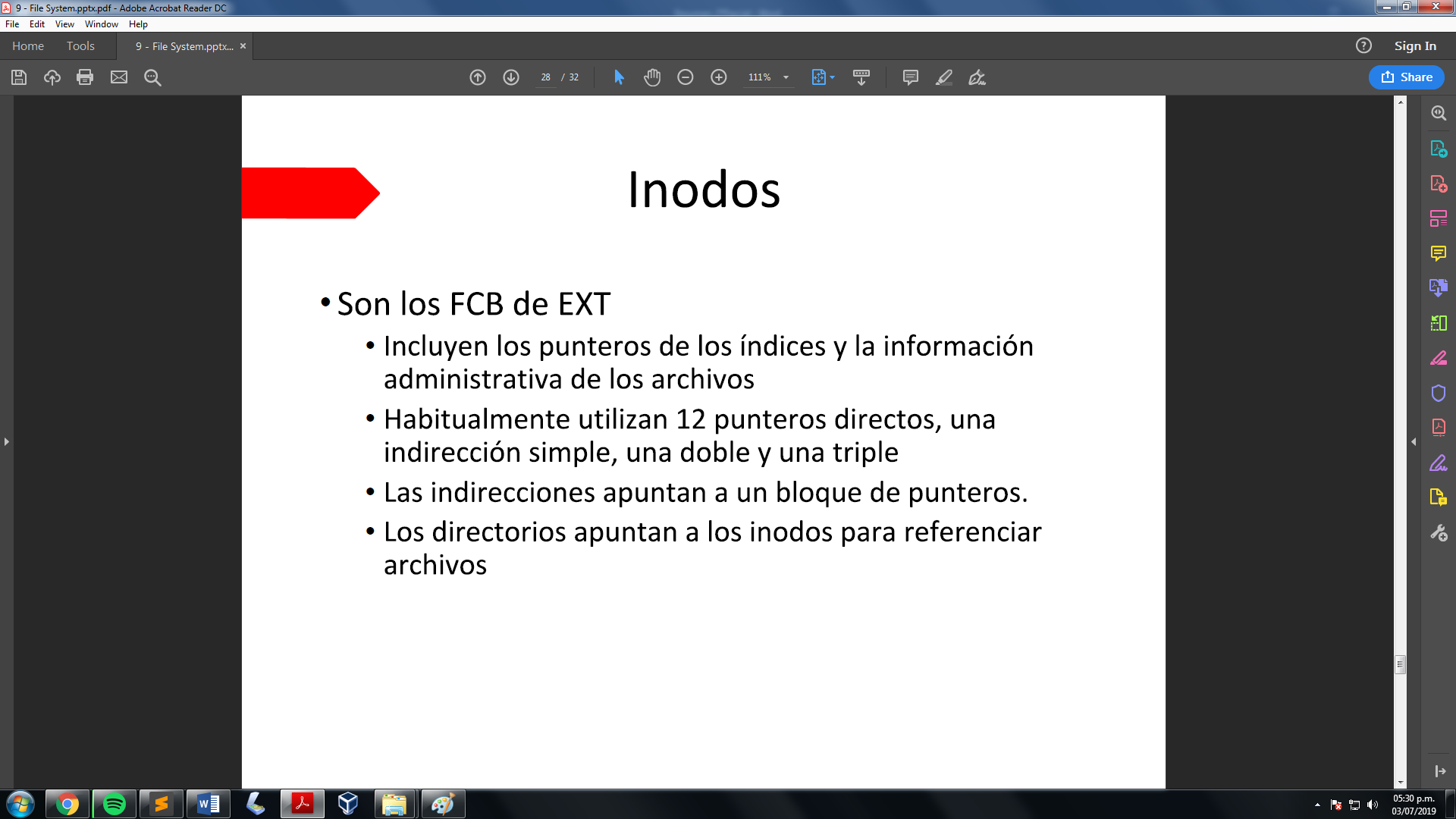
* se utiliza en algunas memorias flash
* versiones 12/16/32/VFAT/ExFAT
* DOS/Windows(XP)

BootSector**||**FAT 1...FAT n**||**Directorio Raiz (RootFolder)**||**Bloques de datos (Cluster)

**EXT**



Inodos



**Accesos Directos (Links)**

* SoftLink / SymbolicLink: para crear un acceso directo se crea un inodo de tipo SoftLink y en el puntero directo (PD) se accede a la ruta del archivo original. O sea, accedo a través de la ruta del archivo original. Para acceder leo todos los bloques del archivo + 1 adicional (la ruta). La desventaja es que sí borro el original ya no puedo acceder al archivo desde el softlink (pierdo la información)
* HardLink: El acceso directo tiene el Id del inodo del original, es mucho más directo. Sí borro el original puedo seguir abriendo el nuevo (acceso directo). Para los inodos apuntados por hardlinks se tiene un contado de hardlinks para saber cuándo borrar el inodo sí se borrarán todos los hardlinks. La desventaja es que sólo puedo direccionar archivos de mi File System y no de otros. En cambio, un SoftLink puede direccionar archivos de otros File System.

**Tareas para agregar información de un archivo (en EXT2)**

1. localizar el archivo
   1. buscar a través de los directorios por medio de la ruta
2. asignarle espacio
   1. sí es necesario se asignan bloques libres al archivo
   2. se modifica el bitmap de bloques y los punteros en el inodo
3. escribir los datos en el nuevo archivo
   1. modificar atributos del archivo (tamaño, fechas)

**FAT vs. UFS (EXT)**

|  |  |  |
| --- | --- | --- |
|  | FAT | UFS |
| Complejidad | Baja | Alta |
| FCB | No | Inodo |
| Espacio libre | No | Bitmap |
| Journaling | No | si |
| Links | No | si |
| Seguridad (Permisos) | No (no posee permisos) | si |

**Swap-out:** agarra la pagina en memoria y va a disco.

**Swap-in:** cargar paginas en memoria (desde disco).

En el disco tendré todo el proceso pero en memoria una parte (por lo general la que voy a utilizar). Cuando no se encuentra la pagina en memoria se produce un “page fault” y se trae la pagina faltante del disco. Para sustituir paginas en memoria se usa el reemplazo local (saco una pagina del mismo proceso que pidió otra pagina). El acceso a disco es muy lento (se trata de que no haya).

Para detectar la anomalía de belady debo contar los page fault del comienzo (cuando cargo las paginas en memoria al iniciar).

La TLB es independiente de los page fault.

Hay un punto en donde agregar mas paginas a mi tabla de paginas no mejora el rendimiento si no que este cae abruptamente llegado a ese punto. Este fenómeno ocurre porque me quedo sin memoria y se empieza a ejecutar mucho el SO. Por lo tanto hay mucho overhead y se la pasa en I/O. Se soluciona de 2 formas:

* Comprar mas memoria
* Menor grado de multiprogramación

Working-Set model:

El rendimiento de los procesos en paginación depende de un conjunto de frames que hace que puedan evitar los fallos de pagina, conocido como working set.

Entradas y salidas I/O:

* Bloqueante: lectura de disco
* No bloqueante (solo se bloquea si el buffer del disco se satura): escritura en disco -> teclado

I/O de carácter: flujo de caracteres no direccionable (teclado)

I/O de bloque: se encarga de direccionar la información, lectura y escritura(disco).

I/O de reloj: manda interrupciones

Las I/O trabajan con N.º de interrupción y N.º de dispositivo para saber donde esta ubicado en el BUS.

Acceso directo a memoria (DMA): transfiere datos desde y hacia la memoria a través del bus del sistema. A veces este obliga al procesador a que se suspenda temporalmente para usar el BUS, a esta técnica se la denomina “robo de ciclos de bus” donde el procesador debe esperar un ciclo de bus mientras la unidad DMA transfiere una palabra. El DMA es mucho mas eficiente que la E/S programada o la dirigida.

Formateo de bajo nivel: borra todos los sectores del disco y los reenumera y me pone SRC para que pueda trabajar. Se usa cuando hay sectores muy bajos del disco que están corruptos (por virus). Borra todo el disco.

MBR (master boot record):

Un registro de arranque principal, conocido también como registro de arranque maestro (por su nombre en inglés **master boot record**, **MBR**) es el primer sector de un dispositivo de almacenamiento de datos, como un disco duro.

Contiene un loader, una tabla de partición de disco y una marca de arranque para que el disco sepa que tiene que arrancar desde ahí.

**Entrada y salida**

Tipos:

Comprensibles por los usuarios, Comprensibles por el sistema, Para comunicación.

Existen muchas diferencias entre dispositivos del mismo tipo. Diferencias:

Velocidad de transferencia, Uso, Complejidad, Unidad de transferencia, Condiciones de error, Acceso aleatorio o secuencial, Compartible o dedicado Lectura, escritura o Lectura/escritura

Existen tecnicas para pasar la informacion desde el SO al dispositivo.

-> E/S programada: La cpu consulta constantemente sí el dispositivo está listo para ser atendido

-> E/S por interrupciones: El dispositivo le envia una interrupción para avisar de una E/S. La cpu ya no se preocupa por la E/S, se mantiene haciendo otras cosas.

-> Acceso directo a memoria DMA: La cpu delega la operacion de E/S y le indica al módulo de DMA:

-Tipo de operacion (Escritura / Lectura).

-Direccion del dispositivo donde operar.

-Cantidad de bytes a operar.

-La ubicación en memoria para operar.

Ya no se encarga de la transferencia.

Cuando el DMA hace algo, le roba la posibilidad al CPU de tener acceso a los demas, pero puede llegar a hacer algún tipo de trabajo mediante su memoria cache y sus registros.

Buffering

Es un espacio de memoria utilizado para la E/S. Lo maneja el SO ya qué es parte de su memoria que permite que todas las E/S pasen por él primero ya sea para leer o escribir. Sirve cómo un adaptador para manejar la información.

* Adapta diferencias de velocidades.
* Adapta unidades de transferencia.

Alamacena datos mientras se transfieren entre dispositivos y aplicaciones.

Buffering de paginas

Cuando se reemplaza una página, está se guarda en el espacio de memoria del SO.

Ventajas:

Una página reemplazada puede estar disponible en memoria, Paginas modificadas se escriben en grupo, Evita el uso del bit de bloqueo de paginas

Planificación de disco

Principal almacenamiento secundario

Memoria virtual

Discos magneticos HDD:

Estructura interna: cuenta con un brazo o cabezal y con platos. Los platos tienen pistas y sectores y caras. LAs cabezas se mueven por el disco para leer o escribir. Al conjunto de sectores se les llama cilindro.

Los sectores suelen ser de 512bytes.

Tiempo de acceso a un sector:

Es la suma de:

* Tiempo de busqueda: Tiempo para posicionar el brazo en la pista deseada.
* Latencia rotacional: tiempo en el qué tarda el sector en posicionarse debajo de la cabeza.
* Tiempo de transferencia

Algoritmos de planificación de disco:

1. FIFO: Atiende por orden de llegada.
2. SSTF: Atiende a la más cercana, produce inancion pero reduce el tiempo de busqueda.
3. SCAN: Atiende de manera ascendente, una vez qué sabe a quien atender no atiende pedidos en el medio, sube hasta el tope y baja para ser atendiendo siempre de manera ascendente.
4. C-SCAN: Tiene una caida abrupta con una perdida de tiempo despreciable, de la pista tope cae a la inicial en 0.
5. LOOK: Igual al SCAN, pero no llega al tope.
6. C-LOOK: Cae justo a donde tiene qué caer, es muy dificil de implementar a nivel HW. Funciona igual qué el LOOK.
7. FSCAN: Se atiende cómo un SCAN, tiene dos colas de pedido, una activa y una pasiva, los pedidos nuevos se agregan a la cola pasiva, cuando todos los pedidos de la cola activa son atendidos, la cola pasiva pasa a ser la activa y la activa la pasiva. no sufre inanición pero lleva más tiempo.
8. N-STEP: Posee colas de pedidos de N o menos pedidos de tamaño fijo, sí N=1 degenera en un FIFO, su forma de atender los pedidos depende del algoritmo q se asigne:
   1. N-STEP-SCAN: los atiende cómo un SCAN
   2. N-STEP-C-SCAN: los atiende cómo un C-SCAN.
9. Sector lógico a CHS: se realiza una divisil para determinar la posicion del cilindro, la cabeza y el sector.

